Vysoká škola báňská – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2020 **Lukáš Hoďák**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání bakalářské práce

Student: **Lukáš Hoďák**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R059 Mobilní technologie

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1.Student vykoná individuální praxi ve firmě: SCOVECO, s.r.o.
2.Struktura závěrečné zprávy:
a.Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta
b.Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti
c.Zvolený postup řešení zadaných úkolů
d.Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe
e.Znalosti či dovednosti scházející studentovi v průběhu odborné praxe
f.Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vedl odbornou praxi studenta

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Zdeňka Chmelíková, Ph.D.**

Konzultant bakalářské práce: Ing. Zdeněk Velart

Datum zadání: 01.09.2019
Datum odevzdání: 30.04.2020

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky 1D16 OSTRAVA

prof. Ing. Miroslav Vozňák, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne: *14. května 2020*

Hodík
.....
podpis studenta

Poděkování

Tímto bych chtěl poděkovat firmě SCOVECO, s.r.o., za možnost vykonání odborné praxe. Dále bych chtěl poděkovat za odbornou pomoc, praktické rady, připomínky a konzultace Ing. Zdeňku Velartovi, Ph.D., a rovněž vedoucí bakalářské práce Ing. Zdeňce Chmelíkové, Ph.D.

Prohlášení zástupce spolupracující právnické nebo fyzické osoby

„Souhlasím se zveřejněním této bakalářské/diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských/magisterských programech VŠB-TU Ostrava."

Dne: *30. dubna 2020*

.....
podpis zástupce

Abstrakt

Tato bakalářská práce popisuje průběh absolvování individuální odborné praxe ve firmě SCOVECO, s.r.o. První část práce je převážně teoretická se zaměřením na použité technologie a nástroje při vývoji systému „Metrics", určený pro sledování a zobrazování metrik pro splnění standardu Automotive SPICE, a také zahrnuje vysvětlení těchto dvou zmíněných pojmů. Následně práce obsahuje praktickou část, ve které je na úvod vysvětleno zadání řešené úlohy, poté její analýza a zmapování funkčnosti. Následuje část implementační, kde je z předchozí analýzy vysvětlen průběh realizace. Závěr této bakalářské práce tvoří souhrn uplatněných, nově získaných zkušeností a rovněž zhodnocení průběhu absolvování odborné praxe.

Klíčová slova

individuálni odborná praxe; SCOVECO, s.r.o.; metrics; Automotive SPICE; Java; JavaServer Faces; PrimeFaces; Liferay; MariaDB

Abstract

This bachelors thesis describes the course of completion of my individual professional practice in the company SCOVECO, s.r.o. The first part of the thesis is mainly theoretical, focusing on technologies and tools used in the development of "Metrics" system designed for monitoring and displaying metrics to meet the Automotive SPICE standard. It also includes an explanation of these two terms. Subsequently, this thesis contains a practical part with description of the asigned task, its analysis and mapping its functionality. Then follows the implementation, where the task is carried out from the prior analysis. The last part of the bachelors thesis, the conclusion, consists of a summary of used and newly acquired skills and experiences and also an evaluation of the course of professional practice.

Key words

professional practice; SCOVECO, s.r.o.; metrics; Automotive SPICE; Java; JavaServer Faces; PrimeFaces; Liferay; MariaDB

Seznam použitých zkratek

Zkratka	Význam
SPICE	Software Process Improvement and Capability Determination
ISO	International Organization for Standardization Process Assessment Model
IEC	International Electrotechnical Commission
BMW	Bayerische Motoren Werke AG
AS	Automotive SPICE
PRM	Process Reference Model
ACQ	Acquisition Process Group
SPL	Supply Process Group
SYS	System Engineering Process Group
SWE	Software Engineering Process Group
SUP	Supporting Process Group
MAN	Management Process Group
PIM	Process Improvement Process Group
REU	Reuse Process Group
PAM	Process Assessment Model
JVM	Java Virtual Machine
WORA	Write once, run anywhere
JSF	JavaServer Faces
XUL	XML User Interface Language
XML	Extensible Markup Language
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
PHP	Hypertext Preprocessor
VCS	Version Control System
SQL	Structured Query Language
API	Application Programming Interface

MIT	Massachusetts Institute of Technology
JDBC	Java Database Connectivity
SSH	Secure Shell
UML	Unified Modeling Language
JAR	Java ARchive
AJAX	Asynchronous JavaScript and XML
CSV	Comma-Separated Values
PDF	Portable Document Format
ID	IDentifier
CSS	Cascading Style Sheets
ORM	Object-rational Mapping
IP	Internet Protocol

Obsah

Úvod ..... - 14 -
1 Teorie ..... - 15 -
1.1 Odborné zaměření firmy, pracovní zařazení ..... - 15 -
1.2 Metriky ..... - 15 -
1.3 Automotive SPICE ..... - 15 -
1.3.1 Process Reference Model (PRM) ..... - 16 -
1.3.2 Process Assessment Model (PAM) ..... - 17 -
2 Použité technologie a nástroje ..... - 18 -
2.1 Java ..... - 18 -
2.2 JavaServer Faces ..... - 18 -
2.2.1 Managed Bean ..... - 18 -
2.3 Eclipse IDE ..... - 19 -
2.4 Git ..... - 19 -
2.4.1 GitLab ..... - 20 -
2.5 Gradle Build Tool ..... - 20 -
2.6 Liferay Portal ..... - 20 -
2.6.1 Service Builder ..... - 20 -
2.7 PrimeFaces ..... - 21 -
2.8 Apache Tomcat ..... - 22 -
2.9 MariaDB ..... - 22 -
2.10 DBeaver ..... - 22 -
3 Praktická část ..... - 23 -
3.1 Zmapování a návrh technologie pro realizaci dané funkčnosti ..... - 23 -
3.2 Analýza a design dané funkčnosti ..... - 23 -
3.3 Implementace ..... - 24 -
3.3.1 Přidání portletu do Liferay portálu ..... - 25 -
3.3.2 Využití Service Builderu ..... - 25 -
3.3.3 Grafické rozhraní webové aplikace ..... - 28 -
3.3.4 Využití Dynamic Query ..... - 35 -

Závěr ..... - 38 -
Použitá literatura ..... - 39 -

Seznam obrázků

Obrázek 1.1 Automotive SPICE — Referenční model procesu [5] ..... - 16 -
Obrázek 2.1 Ukázka PrimeFaces komponenty Timeline ..... - 21 -
Obrázek 3.1 UML diagram systému ..... - 24 -
Obrázek 3.2 Úvodní stránka portálu, vložení portletu ..... - 25 -
Obrázek 3.3 Datový model systému ..... - 26 -
Obrázek 3.4 Grafické rozhraní Service Builderu ..... - 27 -
Obrázek 3.5 Uspořádání tlačítek pro přesměrování ..... - 28 -
Obrázek 3.6 Tabulka zobrazení Automotive SPICE verzí ..... - 29 -
Obrázek 3.7 Formát exportovaných dat tabulky v CSV a PDF souborech ..... - 30 -
Obrázek 3.8 Dialogové okno pro přidání nové metriky ..... - 31 -
Obrázek 3.9 Reprezentace stylizace textu ..... - 32 -
Obrázek 3.10 Dialog smazání záznamu s upozorněním ..... - 33 -
Obrázek 3.11 Zvýraznění řádku s chybějící závislostí ..... - 34 -
Obrázek 3.12 Výběr z vytvořených typů metrik ..... - 34 -
Obrázek 3.13 Reprezentace dat programem DBeaver ..... - 34 -
Obrázek 3.14 Výběr projektových uživatelů a metrik ..... - 36 -

Seznam výpisů zdrojového kódu

Výpis 2.1 Anotace pro managed bean třídu ..... - 18 -
Výpis 2.2 Ukázkový kód použití komponenty Timeline ..... - 21 -
Výpis 3.1 Struktura entity Metric pro Service Builder ..... - 27 -
Výpis 3.2 XHTML struktura tabulky pro zobrazeni ASPICE verzí ..... - 29 -
Výpis 3.3 Zápatí tabulky ASPICE verzí s tlačítky pro manipulaci se záznamy a export dat . - 30 -
Výpis 3.4 Metoda addMetric pro přidání nebo úpravu metrik ..... - 32 -
Výpis 3.5 CSS styl pro zvýraznění řádku ..... - 33 -
Výpis 3.6 Dynamic Query pro získání uživatelů ..... - 36 -
Výpis 3.7 Vytváření nových a odebírání starých projektových uživatelů ..... - 37 -

Úvod

Popularita informačních technologií neustále roste a s ní se z hlediska efektivnosti a praktičnosti zvyšují i nároky na koncový produkt. K sledování kvality software je nutno v průběhu vývoje zaznamenávat různé informace určující jeho kvalitu, které jsou předem definovány standardy. Tato bakalářská práce popisuje průběh mé individuální odborné praxe absolvované ve firmě SCOVECO, s.r.o., jejíž cílem bylo vytvoření systému pro sledování a zobrazování metrik pro splnění standardu Automotive SPICE. Důvody výběru tohoto tématu jsou převážně vyzkoušení uplatnění nabytých informací během studia, a hlavně získání nových praktických zkušeností, mezi které například patří práce na rozsáhlých projektech, využití nových technologií, vývoj v kolektivu apod.

Teoretická část této bakalářské práce se zabývá popisem vybrané společnosti a mého pracovního zařazení v průběhu absolvování odborné praxe. Dále obsahuje vysvětlení pojmů metrik a standardu Automotive SPICE a jeho dvou modelů. Následně zbytek této kapitoly věnuji technologiím a nástrojům, které jsem pro realizaci systému využil.

V praktické části se věnuji popisem postupu, podle kterého se zmíněný systém postupně zpracovával. Vyskytuje se v ní jeho analýza a popis funkčnosti pomocí diagramů. Poté následuje samotná implementace s využitím několika speciálních nástrojů a popis použitých komponent pro tvorbu grafického rozhraní pomocí grafické knihovny.

Závěr bakalářské práce obsahuje shrnutí absolvované odborné praxe s dosaženými výsledky. Dále se v ní nachází souhrn znalostí a dovedností, kterých jsem ze studia v průběhu absolvování uplatnil, a naopak, které při vykonávání scházely. V závěru je také uvedeno celkové zhodnocení odborné praxe.

1 Teorie

1.1 Odborné zaměření firmy, pracovní zařazení

Společnost SCOVECO, s.r.o. se zabývá vývojem informačních a cloudových systémů nebo mobilních řešení na míru. Svým zákazníkům poskytuje služby v rámci vývoje software, ať už z hlediska školení a konzultací se zaměřením na standard Automotive SPICE, analýzy IT řešení, nebo samotným vývojem softwaru. Tento proces zprostředkovává již od prvotních vizí až po jeho úplné nasazení a následnou údržbu [1].

Během absolvování individuální odborné praxe v této firmě jsem se na pozici programátor analytik podílel na analýze, implementaci systému pro sledování metrik pro splnění standardu Automotive SPICE.

1.2 Metriky

Pojmem metrika se v terminologii softwarového inženýrství, tedy v disciplíně zabývající se praktickými problémy při vývoji rozsáhlých softwarových systémů [2], rozumí nástroj, který slouží jako nepřímý ukazatel kvality produktu nebo procesu. Metriky se jako způsob měření používají nejen ve zmíněném softwarovém inženýrství, ale i v jiných odvětvích. Za zmínku například stojí směrovací protokoly v počítačových sítích, kde každý protokol počítá svou metriku odlišným způsobem, a na tomto základě se určuje, která cesta k cílovému zařízení je nejvhodnější. Metriky se tedy používají jako číselné vyjádření při měření a díky těmto známým metrikám je lze porovnávat. Protože kvalita softwaru je abstraktní pojem, který může pro každého znamenat trochu něco odlišného, bylo nutné, aby se tato kvalita dala reprezentovat jednotným způsobem. Díky tomu lze sledovat jednotlivé fáze vývoje a porovnávat je mezi sebou, jestli se například kvalita vyvíjeného software zvyšuje, či naopak snižuje [3]. Celková kvalita softwaru se odvíjí od jednotlivých kvalit dílčích částí, ať už z hlediska osob (zákazník, dodavatel), které s daným systémem přichází do styku, nebo z hlediska procesu vývoje, produktu, zdrojových prostředků a dalších. Aby se pokaždé tyto jednotlivé metriky, které tvoří základ pro sledování kvality, nemusely vždy při vývoji nového software znovu definovat a ušetřil se tím čas a práce, začalo se tyto metriky standardizovat. Jedním ze standardů zaměřených na posuzování kvality je i standard Automotive SPICE, který je detailněji popsán v následující kapitole 1.3.

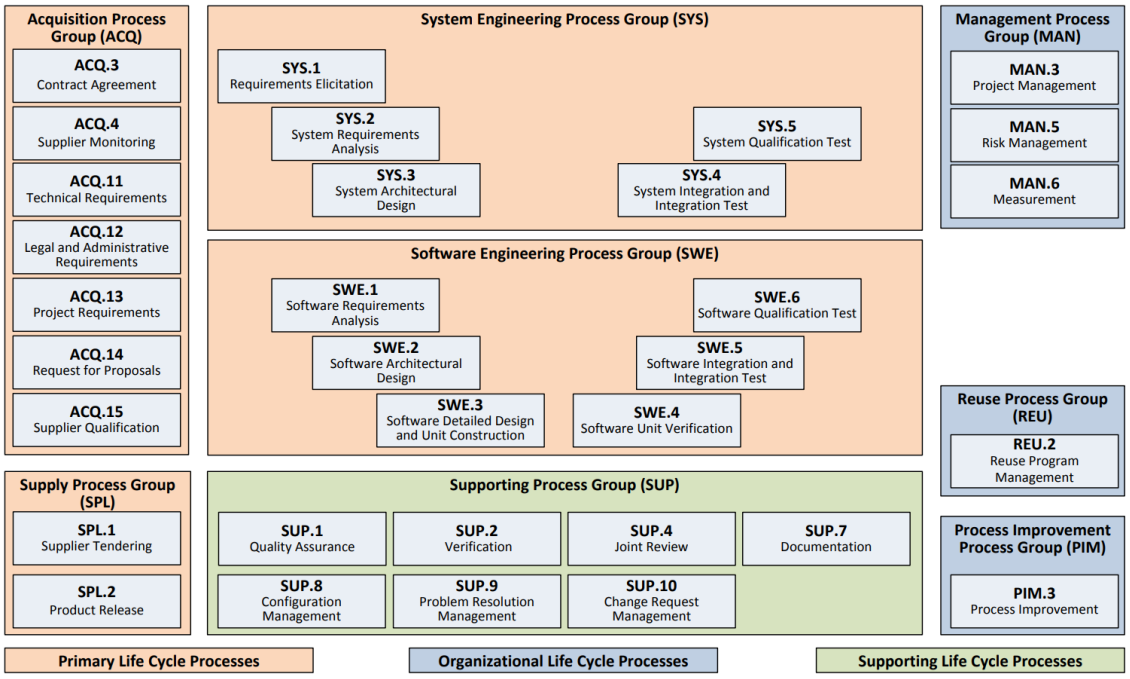
1.3 Automotive SPICE

Automotive SPICE (Software Process Improvement and Capability Determination) je mezinárodní standard pro posuzovaní procesů zaměřených na vývoj software. Tento standard se převážně používá v automobilovém průmyslu a jeho služeb využívají nejen přední německé automobilové závody (BMW, Audi, Volkswagen, Porsche a další), ale i jiné mezinárodní společnosti. Tento standard (často označován jako framework) spadá pod Mezinárodní organizaci pro normalizaci ISO (International Organization for Standardization) a Mezinárodní elektrotechnickou komisi IEC (International Electrotechnical Commission), konkrétně označen pod normou ISO/IEC 15504 [4].

Dříve zmíněný pojem hodnocení procesu je definován jako způsob ohodnocení jednotlivých procesů organizační jednotky v závislosti na referenčním modelu. K hodnocení procesu využívá Automotive SPICE dva modely [5].

1.3.1 Process Reference Model (PRM)

V referenčním modelu procesu jsou definovány všechny Automotive SPICE procesy, které lze použít při vývoji software a vestavěných automobilových systémů. Jedná se o model seskupující definice procesů a jejich životních cyklů popsaných se zaměřením na jejich účelovost a výsledky.



Obrázek 1.1 Automotive SPICE — Referenční model procesu [5]

Primární životní cyklus se zabývá procesy, k jejichž využití může dojít při styku zákazníka s dodavatelem, tedy při obdržení a dodávání produktu. Primární cyklus se ještě dále člení do následujících skupin podle procesů (process groups):

- *Acquisition process group (ACQ)* - využívají se při aktivitách souvisejících s obdržením produktu nebo služeb
- *Supply process group (SPL)* - procesy prováděné dodavatelem pro poskytnutí produktu nebo služeb
- *System Engineering process group (SYS)* - procesy zaměřené na řízení zákaznických a interních požadavků
- *Software Engineering process group (SWE)* - procesy zabývající se softwarovými požadavky odvozených od systémových požadavků odpovídajícím architektuře a návrhu software

Kategorie životnosti podpůrných procesů má na starost procesy, které mohou být kdykoliv použity ostatními procesy, nezávisle na životním cyklu. Patří zde kategorie SUP.

Třetí, a zároveň poslední, organizační životní cyklus tvoří procesy vývoje, produktu a zdrojových prostředků. Podobně jako v případě u primárního, se i tento cyklus dále rozvíjí na další skupiny podle procesů, které zahrnují:

- *Management process group (MAN)* - procesy pro správu jakéhokoliv projektu
- *Process Improvement process group (PIM)* - obsahuje pouze jeden proces s praktikami zaměřenými na vylepšování
- *Reuse process group (REU)* - skupinu tvoří pouze jeden proces zaměřený na využití znovupoužitelnosti programů organizace [6]

1.3.2 Process Assessment Model (PAM)

Model hodnocení procesů je určen k použití při vývoji zabudovaných automobilových systémů, konkrétně při provádění zhodnocení schopnosti procesu vývoje těchto systémů. V rámci použití PAM se hovoří o několika indikátorech, které se při jeho použití zohledňují. Tyto indikátory se rovněž používají při implementaci programu pro vylepšení procesu kvality. Princip fungování PAM je vyobrazen pomocí dvoudimenzionálního frameworku, kdy jedna dimenze se skládá z procesů definovaných referenčním modelem. Druhá dimenze je tvořena z jednotlivých úrovní schopností, které se ještě dále rozdělují na atributy procesu. Ty poskytují měřitelné vlastnosti způsobilosti procesu [7].

S narůstajícím využitím a integrací softwaru při každé části vývoje nového software a jeho neustálým rozvojem je pro zajištění kvality žádoucí, aby tato pravidla, definovaná standardem Automotive SPICE, byla v co největší míře splněna.

2 Použité technologie a nástroje

Pro splnění jednotlivých úkolů odborné praxe bylo zapotřebí využít mnoha softwarových nástrojů a technologií. S některými z nich jsem už měl během studia předešlé zkušenosti, avšak byly mezi nimi i některé, které byly pro mě nové. Tato kapitola se zabývá jejich popisem a vysvětlením.

2.1 Java

Java je objektově orientovaný programovací jazyk založený na rozložení kódu do tříd. Její tvůrci, americká firma Sun Microsystems, se snažili vytvořit jednoduchý, ale přesto přenositelný programovací jazyk. Toho dosáhli tím, že oproti ostatním programovacím jazykům Java nevytváří skutečný strojový kód, ale vytváří svůj tzv. „bytecode". Ten není závislý na architektuře a lze ho tedy spustit na libovolném zařízení, které má dostupný virtuální stroj Java (JVM) sloužící jako interpret tohoto bytecode [8]. Tato vlastnost bývá označována jako „Write Once Run Anywhere (WORA)" [9]. Java podporuje „just-in-time" překlad, díky kterému se překládají jen ty části kódu, které jsou v danou chvíli nutné pro běh programu [10].

2.2 JavaServer Faces

JavaServer Faces, zkráceně JSF, je framework (platforma pro vývoj softwarových aplikací) s uživatelským grafickým rozhraním založeným na komponentách. JSF je součástí platformy Jave Enterprise Edition, z čehož vyplývá, že pro použití tohoto frameworku není nutná žádná dodatečná instalace knihoven. Princip fungování spočívá v oddělení části s definovaným uživatelským rozhraním od části s její logikou. Díky tomu umožňuje JSF použít různé značkovací jazyky, např. XUL a HTML [11].

2.2.1 Managed Bean

K přidání logiky a chování komponent na uživatelském rozhraní využívá JavaServer Faces tzv. „managed bean". Pro definování Java třídy jako managed bean je nutno přidat následující anotaci pod zahrnuté knihovny.

```
@ManagedBean(name = "mainView")
```

Výpis 2.1 Anotace pro managed bean třídu

Takto se daná třída označí pod názvem „mainView" a pomocí tohoto identifikátoru lze přistupovat k vytvořeným metodám a proměnným této třídy. Avšak pro práci s proměnnými platí, že musí mít vytvořené své odpovídající „get" a „set" metody, aby se jejich hodnoty mohly číst a zapisovat. Za označením, určující že se jedná o managed bean, následuje anotace rozsahu (scope) patřičného beanu. Tento rozsah se dá chápat jako jeho životnost. Nabízených rozsahů pro managed beany je několik, liší se mezi sebou v okamžiku a podnětu jejich vzniku a zániku. Tento jejich životní cyklus se může vázat na různé rozsahy a mezi nejčastěji používané patří:

- ApplicationScoped – životnost beanu je vázána na jeho webovou aplikaci. Vytvoří se buď při prvním HTTP požadavku, nebo pokud je navíc u deklarace ManagedBeanu nastavena vlastnost „eager" na true, tak při prvotním načtení aplikace. S ukončením aplikace končí i bean.
- RequestScoped – bean existuje po celou dobu, stejně tak jako HTTP požadavky a odpovědi. K vytvoření dojde při HTTP požadavku a zaniká při odpovědi související s tímto požadavkem. Tento scope je jako výchozí hodnota vybrán automaticky, pokud není jinak definováno.
- SessionScoped – bean existuje po celou dobu od vytvoření HTTP spojení (session) do zániku této session
- ViewScoped – v tomto případě bean existuje, zatímco uživatel provádí jakoukoliv interakci se stejným JSF v rámci jednoho webového okna/záložky (view). Stejně jako u předchozího se vytvoří při HTTP požadavku a zaniká, jakmile uživatel přejde na jiné view [12].

2.3 Eclipse IDE

Eclipse je v současnosti jedním z nejpoužívanějších integrovaných vývojových prostředí pro programovací jazyk Java. První verze byla vydána roku 2001 a od té doby je prostředí Eclipse neustále vyvíjeno. Kromě primárního zaměření na Javu také podporuje vývoj v jiných programovacích jazycích, jako jsou například C/C++, JavaScript/TypeScript, PHP a další. Samotný program si lze do značné míry přizpůsobit podle svých potřeb a komfortu. Na webových stránkách i v aplikaci nabízí tvůrci tzv. Eclipse Marketplace, kde velké společnosti, ale i jednotlivci, mohou umísťovat různé nástroje, které primárně slouží pro zvýšení efektivity a přehlednosti při využívání tohoto IDE. Dále je zde k dispozici několik perspektiv, což jsou sbírky pohledů přizpůsobených na danou vykonávanou činnost, mezi kterými lze jednoduše přecházet (aktivní může být v jeden moment pouze jedna), obsahujících nástroje pro přehlednou práci podle daného výběru. S instalací vývojového prostředí Eclipse se nahraje už několik přednastavených perspektiv, ale každou si lze upravit nebo vytvořit svou novou [13]. Využité technologie Git a Liferay Portal, o kterých je jednáno v práci později, mají v tomto vývojovém prostředí také své vlastní pohledy.

2.4 Git

Git je systém určený pro verzování při vyvíjení software. Spolu s nejpoužívanějšími řešeními, jako jsou například Beanstalk, Bitbucket, Microsoft Team Foundation Server apod., je Git označován anglickou zkratkou jako VCS (Version Control System). Pojmem verzování se rozumí efektivní sledování a práce se změnami provedenými při postupném vývoji software. Při používání VCS má každý podílející se uživatel lokálně uloženou kompletní kopii. Tomuto procesu se říká klonování (cloning). Po provedení změn se aktuální upravený stav uloží příkazem „commit". Aby se dané úpravy mohly sdílet s ostatními uživateli pracujícím na stejném projektu, je nutno tyto lokální změny nahrát do hlavního vzdáleného repozitáře (master), tato akce se označuje jako „push". Pokud v tomto mezičase jiný uživatel také provedl push, tak se obsah

vzdáleného sdíleného repozitáře neshoduje s tím lokálním a musí se provést operace „pull". Poté je nutno se vypořádat s provedenými změnami, kolizemi apod. a nahrát svou verzi [14]. Tímto způsobem Git značně zjednodušuje kolektivní spolupráci mezi více lidmi podílejícími se na stejném vývoji a umožňuje tím také budovat strukturu vývoje software [15][16].

2.4.1 GitLab

Firma SCOVECO, s.r.o., u které jsem absolvoval odbornou bakalářskou praxi, používá pro řešení verzování webový Git repozitář označený jako GitLab, který po přihlášení umožňuje sledovat jednotlivé zápisy a změny provedené ve vyvíjené aplikaci. Navíc kromě bezplatné verze nabízí i spoustu dalších placených, které mimo jiné obsahují možnost změnu přístupových práv pro podílející se spolupracovníky a spoustu dalších [17].

2.5 Gradle Build Tool

Gradle Build Tool je open source nástroj pro automatizaci sestavování programu, který je zaměřen především na flexibilitu a výkonnost. Kroky, které Gradle postupně vykonává jsou:

- Správa dependencies (závislosti jednotlivých komponent)
- Kompilování zdrojového kódu do spustitelného kódu
- Vytváření spustitelného souboru

Oproti konceptům Apache Ant a Apache Maven, ze kterých Gradle vychází, používá pro sestavovací skripty namísto XML formulářů doménově specifický jazyk Groovy nebo rovněž Kotlin [18].

2.6 Liferay Portal

Liferay Portal je bezplatný portál, který vznikl v roce 2000 a je založen na programovacím jazyce Java (viz 2.1). Díky zabudovaným nástrojům umožňuje snadné a rychlé vytvoření svého vlastní webového portálu (portletu) s použitím předem vytvořeného systému pro správu uživatelů a jejich rolí, autentizaci atp. Mimo jiné mezi další nejdůležitější a nejpoužívanější nástroje, kterými portál Liferay disponuje, jsou nástroje pro správu dat, procesů, aplikací, poskytování služeb [19].

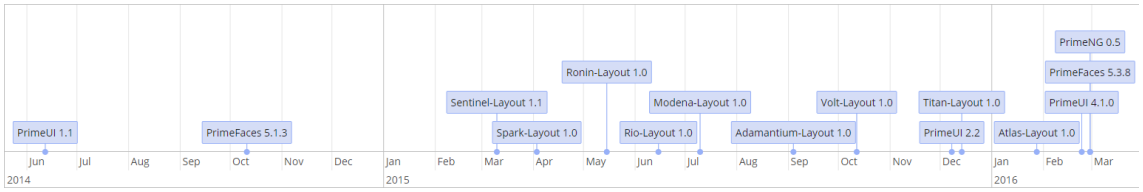
2.6.1 Service Builder

Jedná se o nástroj určený pro vygenerování kódu, založeného na předdefinovaném datovém modelu, který je součástí Liferay Portalu. Zdrojový soubor, ze kterého generátor vychází, je popsán v příslušném XML souboru nazývaném „service.xml". V tomto souboru se nachází struktura vygenerovaných tříd – jejich atributy, základní metody a také jejich vzájemné návaznosti. K tvoření obsahu tohoto souboru slouží buď vytvořené uživatelské rozhraní nebo čistě textové rozhraní. Service Builder takto umožňuje pomyslně oddělit databázovou vrstvu od vrstvy s vlastní implementací a ušetřit čas s návrhem kódu pro práci s databází. Umožňuje také manipulaci s uloženými daty s použitím Java syntaxe a tím se značně omezuje potřebná znalost syntaxe SQL jazyka [20].

Každý vygenerovaný objektový model, který se anglicky označuje jako entita, má svou vlastní třídu implementace modelu. Dále také obsahuje místní nebo i vzdálené třídy implementace. Místní služby jsou zodpovědné za získávání a ukládání dat, vzdálené poté mají na starost převážně kontrolu přístupových práv, protože jsou určeny pro přístup odkudkoliv z internetu.

2.7 PrimeFaces

Knihovna PrimeFaces poskytuje komponenty pro vytváření grafického rozhraní webové aplikace. V základní verzi se jedná o projekt s volně přístupným zdrojovým kódem na základě MIT licence [21], díky čemu lze dohledat zdrojový kód a všechny poskytované komponenty se dají jednoduše přizpůsobit podle vlastních potřeb pomocí definovaných vlastností. Knihovna poskytuje přes sto prvků uživatelského rozhraní s již zabudovanými přednastavenými styly a s předdefinovaným chováním pro jednoduchou implementaci. Každý prvek má dostupnou svou detailní dokumentaci se seznamem všech parametrů, jejich možnostmi a rovněž i názorné příklady použití dané komponenty [22]. Pro názornost jsem vybral původní ukázku pro komponentu „Timeline" určenou pro zobrazení údajů na časové ose (Obrázek 2.1). Zdrojový kód pro implementaci této časové osy se nachází na obrázku níže (Výpis 2.2) [23].



Obrázek 2.1 Ukázka PrimeFaces komponenty Timeline

```
<p:growl id="growl" showSummary="true" showDetail="true">
    <p:autoUpdate />
</p:growl>

<p:timeline id="timeline" value="#{basicTimelineView.model}" height="250px"
        selectable="#{basicTimelineView.selectable}"
        zoomable="#{basicTimelineView.zoomable}"
        moveable="#{basicTimelineView.moveable}"
        stackEvents="#{basicTimelineView.stackEvents}"
        axisOnTop="#{basicTimelineView.axisOnTop}"
        eventStyle="#{basicTimelineView.eventStyle}"
        showCurrentTime="#{basicTimelineView.showCurrentTime}">

    <p:ajax event="select" listener="#{basicTimelineView.onSelect}"/>

</p:timeline>
```

Výpis 2.2 Ukázkový kód použití komponenty Timeline

Kromě neplacené verze nabízí PrimeFaces i roční předplatnou službu ELITE, která zpřístupňuje nové funkce a zahrnuje i různé grafické šablony, které jsou bez použití této služby placené. Použití šablon vede k jednotnému stylu komponent. Balíček PRO je zaměřen na poskytování podpory uživatelům a slibuje asistenci s délkou odezvy do jednoho dne nebo i řešení problému s využitím vzdáleného přístupu k počítači [24].

2.8 Apache Tomcat

Projekt Apache Tomcat je jedním z nejpoužívanějších webových kontejnerů pro implementaci Java Servletů, JavaServer Pages, Java Expression Language a Java WebSocket specifikací. Webový kontejner je prostředí, ve kterém běží webové komponenty. Má na starost jejich správu a zodpovídá za jejich životnost, souběžnost, bezpečnost a taky jim umožňuje přistupovat k API [25]. Díky jeho robustnosti a jednoduchosti použití se používá jako řešení pro malé místní servery, ale i pro velká podniková řešení [26]. Tomcat spadá pod licenci Apache License 2 používanou pro distribuci software [27].

2.9 MariaDB

MariaDB nabízí bezplatný systém pro správu relačních databází. Na vývoji se podílí původní vývojáři populárního alternativního řešení databázového systému MySQL. Tyto dvě technologie jsou díky tomu spolu kompatibilní a přechod z jedné na druhou by měl být bezproblémový [28]. Mezi hlavní pilíře, na kterých MariaDB staví své řešení databázové struktury, patří zejména nízké finanční nároky na provoz, s ním spojenou správu a údržbu, dále vysoká rychlost zpracování požadavků využitím většího počtu vláken, optimalizace přístupu k disku, poddotazům a dalším. Důraz také klade na bezpečnost dat svých klientů, v případě narušení pracuje na okamžitém vydání záplat [29]. Právě díky těmto nabízeným službám se jedná v dnešní době o jedno z nejpoužívanějších řešení relačních databází pro vzdálené cloud servery a Linuxové distribuce [30].

2.10 DBeaver

DBeaver je volně dostupný nástroj sloužící pro správu databázových systémů. Je určen především pro vývojáře, SQL programátory, administrátory a analytiky. DBeaver dokáže spolupracovat s jakýmkoliv typem databáze za předpokladu, že využívá JDBC driver [31]. Jedná se o komponentu umožňující připojení k databázi pomocí standardních API dostupných platformě Java. DBeaver nabízí prostředky pro práci s uloženými daty jako s běžnou tabulkou, dále umožňuje ze záznamů vytvářet analytické zprávy, exportování dat do několika podporovaných typů souborů. Má také v sobě zabudovaný pokročilý SQL editor, nástroje pro sledování přístupu k databázi a mnoho dalšího [32].

3 Praktická část

Zadaným úkolem individuální odborné praxe bylo vytvořit systém, který sleduje a zobrazuje metriky pro splnění standardu ASPICE. Tento rozsáhlý problém lze rozdělit do dílčích částí, které jsou popsány dále v této kapitole. Jedná se o zmapování a návrh technologie, analýza funkčnosti systému a jeho design, a na závěr samotná implementace.

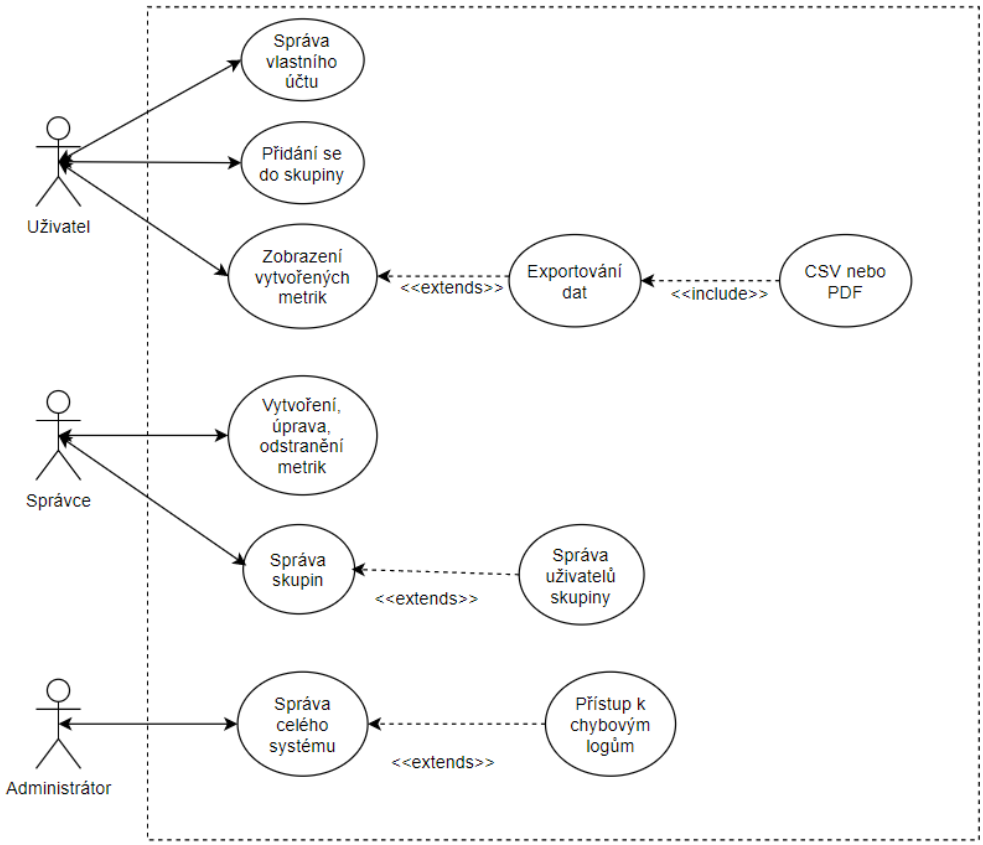
3.1 Zmapování a návrh technologie pro realizaci dané funkčnosti

Po nastoupení na začátku zimního semestru do firemní kanceláře společnosti SCOVECO, s.r.o. jsem se navíc k informacím získaným při návštěvě firemního zázemí před výběrem tématu bakalářské práce dozvěděl konkrétnější pokyny o průběhu odborné praxe. Byl jsem zařazen do dvoučlenného týmu s dalším studentem, který taktéž vykonával bakalářskou praxi s tím, že na základních částech vývoje jsme měli spolupracovat, a později aby každý z nás pracoval samostatně. Ze začátku bylo nutné, abychom si nainstalovali všechen potřebný software, který byl pro realizaci úkolu potřebný. Zvoleným integrovaným vývojovým prostředím byl Eclipse (viz 2.3) Enterprise edice, který jsem během svého studia už používal, a tak jsem s jeho používáním měl předchozí zkušenosti. Dále nám byly zřízeny přístupové účty pro webový repozitář GitLab (viz 2.4.1), firemní cloud úložiště Nextcloud a další. Následovalo vytvoření SSH klíčů pro propojení zmíněného Git účtu s vývojovým prostředím, abychom průběžně mohli provedené změny nahrávat do webového repozitáře. Pro vytvoření a provoz místního databázového systému bylo zvoleno použití programu MariaDB (viz 2.9), pro zobrazování uložených záznamů a práci s nimi jsem použil program DBeaver (viz 2.10). Během instalace a nastavení všech potřebných aplikací jsem studoval problematiku metrik a využití standardů pro zlepšení procesu vývoje software. Poté nám bylo zadáno, abychom se s vybranými technologiemi nejprve seznámili a vyzkoušeli jejich použití. Jednalo se například o využití nástrojů Liferay portálu, vytváření uživatelského rozhraní pomocí komponent z grafické knihovny PrimeFaces, ukládání dat do databáze a práci s nimi atd. Seznámení se s firemním prostředním, instalace a nastavení potřebných nástrojů a následně jejich vyzkoušení trvalo přibližně čtvrtinu odborné praxe.

3.2 Analýza a design dané funkčnosti

Prvním a zároveň velmi důležitým krokem bylo uvědomit si, jak postupně bude probíhat vytváření samotného systému. S kolegou jsme se každý samostatně snažili vymyslet, jak by tento systém měl fungovat, a poté jsme naše nápady společně s konzultantem zhodnotili a sjednotili. Základní vize byla taková, aby webová aplikace umožňovala uživatelům vytvoření metrik a interakci s nimi. Uživatelé se tedy musí nejprve pro práci se systémem zaregistrovat. Po registraci má přihlášený uživatel přístup k nastavení svého účtu dovolující mu změnit své osobní údaje. Uživatel si může založit svůj projekt a automaticky se po jeho vytvoření stane jeho správcem. V této roli může přidávat ostatní uživatele systému do projektu, přičemž dochází k vytvoření konkrétních projektových uživatelů. Hlavní role správce spočívá v tom, že může vybírat z předem vytvořených metrik a přidávat je do projektu. On i ostatní uživatele, kteří jsou

součástí projektu, si můžou zobrazit detail metriky (snapshot). Metriky a na nich další závislé části vytváří administrátor systému. Takto popsanou funkcionalitu systému jsme použitím nástrojů webové stránky pro tvorbu diagramů (https://app.diagrams.net/) převedli do UML diagramu (Obrázek 3.1). UML je speciální jazyk zaměřený na grafické modelování businessů nebo analýzu, design a implementaci softwarových systémů [33]. S tvorbou těchto diagramů jsem už měl díky předmětu Úvod do softwarového inženýrství částečnou předchozí zkušenost, kterou jsem mohl při tvorbě UML diagramu pro tento systém uplatnit. Analýzou a designem funkčností systému jsem strávil necelou druhou čtvrtinu času.



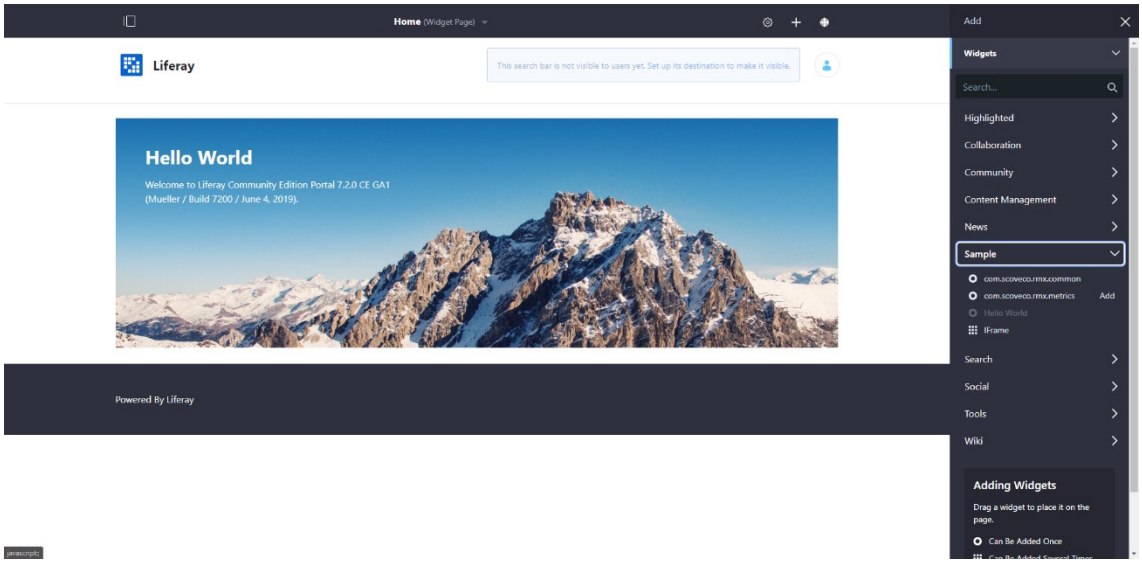
Obrázek 3.1 UML diagram systému

3.3 Implementace

Ruční implementace celého systému, popsaného v předchozí kapitole 3.2, by byla velmi náročná a pracná. K usnadnění práce, a zároveň k propracovanému řešení, jsme využili služeb Liferay portálu, který je zmíněn v kapitole 2.6. Detailní popis použití jeho funkcí pro realizaci projektu na základě předchozího popisu je popsaný v následujících kapitolách zabývajících se implementací systému. Jednalo se o časově nejnáročnější část, na které se pracovalo až do samotného konce odborné praxe, jelikož bylo nutno vytvořit pro většinu entit z datového modelu (Obrázek 3.3) samostatnou stránku, která měla obsahovat potřebné prvky a splňovat své funkce popsané v teoretické části této bakalářské práce.

3.3.1 Přidání portletu do Liferay portálu

Po zprovoznění portálu následovalo přidání zdrojů portletu do vytvořeného lokálního Apache Tomcat serveru (viz 2.7) a jeho spuštění. Portlet je označení webových komponent na základě Java Portlet specifikace, která vznikla za účelem interoperability — umožnění komunikace systému mezi portály a portlety. Touto komunikací se rozumí zpracovávání požadavků a vytváření odpovědí [34]. Portlety ale tvoří pouze část webové stránky, může jich tedy být na jedné stránce několik [35]. Při prvotním spuštění probíhá několik inicializačních operací, jako například kontrola a stahování potřebných JAR souborů, vytváření databázových tabulek, aplikování nastavených vlastností z konfigurace serveru. Výchozí port, na kterém server pracuje, je port 8080, ale ten lze v případě nutnosti změnit, například při provozování jiné služby na zmíněném portu. Po úspěšném spuštění lze přejít ve webovém prohlížeči na adresu http://localhost:8080/ a zobrazí se úvodní stránka Liferay portálu s krátkým průvodcem nastavení. V tomto průvodci se nachází potvrzení přijmutí podmínek použití, a následně se vytváří účet správce portálu, nastavuje se pro něho kontrolní otázka v případě zapomenutí hesla a další. Posledním krokem pro zprovoznění projektu, na takto vytvořeném a nastaveném portálu, je jeho přidání. To lze udělat pomocí kontextového menu s názvem „Add" v pravé horní části obrazovky. Následně je nutno zvolit položku „Widget" a konečně v podmenu „Sample" najít svůj projekt a kliknutím ho přidat. K zobrazení obsahu je uživatel ještě vyzván, aby stránku obnovil. Takto se podle výchozího nastavení zobrazí obsah souboru s názvem „view.xhtml".

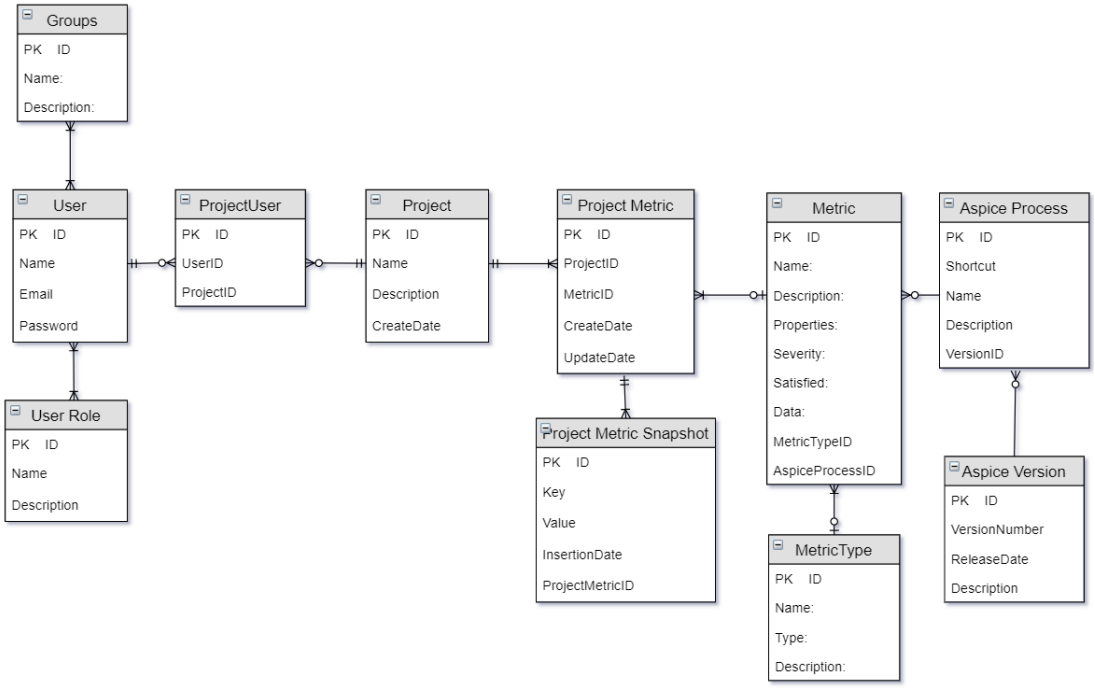


Obrázek 3.2 Úvodní stránka portálu, vložení portletu

3.3.2 Využití Service Builderu

Dalším krokem v implementaci systému bylo přenesení slovního popisu systému do kódu programovacího jazyka Java. K tomu jsme použili další z nástrojů portálu Liferay. Jedná se o Service Builder, jehož detailní popis se nachází v kapitole 2.6.1. Jak už bylo v teoretické části zmíněno, tento nástroj potřebuje pro vygenerování kódu vstupní data zadaná ve formátu XML. Bylo nám doporučeno, abychom z UML diagramu (Obrázek 3.1) vytvořili datový model tvořen

entitami, které reprezentují jednotlivé částí systém. Entity mají dále mít své atributy, které ji definují a jsou mezi sebou navzájem propojeny vazbami v závislosti na jejich vztahu. Datový model (Obrázek 3.3) jsme vytvořili pomocí stejného webového nástroje, který jsme použili i pro tvorbu UML diagramu. Entity jsou v tomto diagramu znázorněny pomocí obdélníků, které mají v záhlaví svůj název, a pod oddělující čarou se vyskytují jejich atributy. Mezi entitami jsou znázorněny jejich jednotlivé vazby a závislosti.

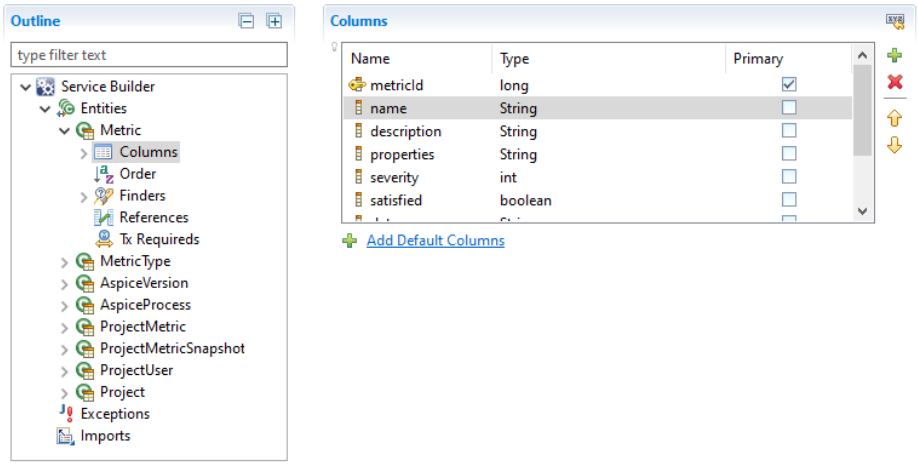


Obrázek 3.3 Datový model systému

Takto vytvořený datový model slouží jako grafické znázornění následujícího popisu: jeden ASPICE proces musí být založený pouze na jedné konkrétní verzi, ale současně z jedné verze může vycházet více procesů. Na tomto principu je založen i vztah mezi metrikami a s nimi svázanými ASPICE procesy. Vytvořená metrika vychází pouze z jednoho typu a zpětně může být jeden typ metriky použit pro vícero metrik. V jednom projektu lze sledovat větší množství metrik, a zároveň jedna konkrétní metrika může být součástí několika projektů. Touto vazbou vznikla mezi nimi vazební entita, která přebírá údaje z obou vycházejících entit, a navíc má jako atributy uvedeny ještě data týkající se jejího vytvoření, aktualizace a údaje o samotných datech. Entita „Project Metric Snapshot" slouží k zachycení konkrétního stavu projektu.

Potřebný XML soubor se dá vytvořit buď pomocí nabízeného uživatelského rozhraní (Obrázek 3.4), do kterého se hodnoty z datového modelu postupně vyplní, nebo lze využít textového režimu, jako tomu bylo i v našem případě. Definice entit se pomocí textu vkládá do párových tagů. V uvozovacím tagu se určují parametry, kterými jsou jméno entity a speciální parametry local-service a remote-service. První z nich určuje vytvoření tříd pro využití stejným Java virtuální strojem (JVM), zatímco druhý slouží pro nastavení, zda se mají třídy vytvářet pro přístup z vnějšího prostředí. Při pojmenovávání entit a sloupců jsem z hlediska konvence

dodržoval tzv. „camelCase" anotaci, která definuje způsob psaní složených slov. Každé slovo, s výjimkou prvního, začíná velkým písmenem a slova nejsou mezi sebou nějak oddělena [36]. Za uvozovacím tagem následuje definice atributů – sloupců. Každý sloupec musí mít přiřazené jméno a odpovídající datový typ. Pro použití jsou k dispozici standardní nativní datové typy i specifické typy pro ukládání dat do databáze, např. BigInt, BigDecimal, LongText aj. Jeden ze sloupců musí být vybrán jako primární, a jeho klíč je určen jako zastupující pro celou entitu. To značí nastavení parametru „primary". Liferay portál kromě základních možností pro získání objektů z databáze poskytuje i možnost pro vytvoření dodatečných metod pro práci s databází na základě specifikovaných sloupců. Jedná se o tzv. „finder" metody, pro jejichž použití je nutno určit návratovou hodnotu a název sloupce, pro který se metody mají vytvořit [37]. Podle těchto pravidel bylo nutno takto vytvořit celý datový model (Obrázek 3.3) a následovalo použití zabudované funkce „build-service" v integrovaném vývojovém prostředí Eclipse a všechny potřebné třídy pro chod byly po krátké chvíli vytvořeny.



Obrázek 3.4 Grafické rozhraní Service Builderu

```
<service-builder dependency-injector="ds"
    package-path="com.scoveco.rmx.metrics.servicebuilder">
    <namespace>metrics</namespace>
    <entity local-service="true" remote-service="false"
        name="Metric">
        <column name="metricId" primary="true" type="long"></column>
        <column name="name" type="String"></column>
        <column name="description" type="String"></column>
        <column name="properties" type="String"></column>
        <column name="severity" type="int"></column>
        <column name="satisfied" type="boolean"></column>
        <column name="data" type="String"></column>
        <column name="metricTypeId" type="long" />
        <column name="aspiceProcessId" type="long" />
        <finder return-type="Collection" name="MetricType">
            <finder-column name="metricTypeId" />
        </finder>
        <finder return-type="Collection" name="AspiceProcess">
            <finder-column name="aspiceProcessId" />
        </finder>
    </entity>
```

Výpis 3.1 Struktura entity Metric pro Service Builder

3.3.3 Grafické rozhraní webové aplikace

Po zprovoznění webové aplikace na serveru a použití nástroje pro generování tříd jsme si museli rozdělit práci, promyslet vzhled a strukturu systému, jaké prvky použijeme, jejich umístění apod. Jednalo se pro mě o nové záležitosti, protože pokud bych opomenul pár menších, převážně školních projektů, tak jsem se osobně s tvorbou grafického rozhraní předtím nezabýval. Musel jsem se seznámit s knihovnou PrimeFaces (viz 2.7), a to jak s její širokou škálou nabízených komponent, tak i s jejich použitím a následnou úpravou. Bylo nám zadáno, abychom pro každou potřebnou entitu z datového modelu vytvořili svou vlastní stránku (view). V tomto view by měla být zobrazena data uložená v databázi, poskytovanou službou MariaDB (viz 2.9), zpracována formou přehledné tabulky. Při popisování jednotlivých komponent a jejich chování, pokud nebude jinak uvedeno, se budu v rámci této práce nejčastěji zaměřovat na popis zpracování entity Metric, která je ze všech obsahově nejnáročnější, a řešení ostatních tabulek je tedy velmi podobné.

3.3.3.1 Úvodní stránka

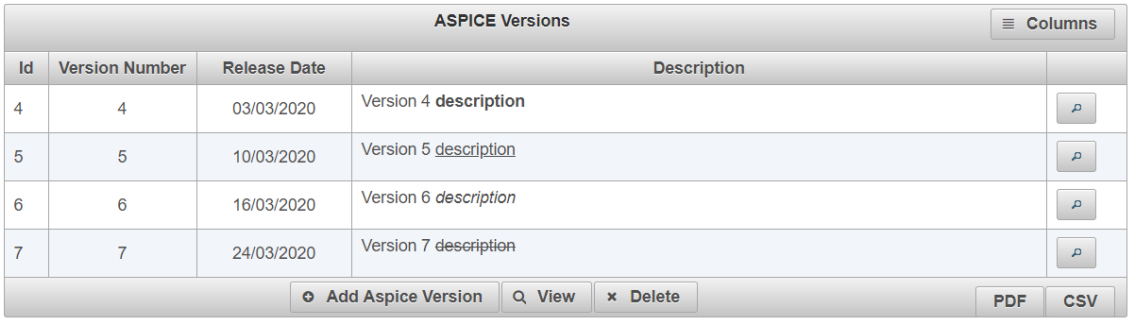
Na úvodní stránku přidaného portletu, která se po připojení na službu objeví, jsme postupně přidávali pro každou zpracovanou entitu komponentu „linkButton". Ta, jak už název napovídá, je reprezentována tlačítkem, po jehož stisknutí dojde k přesměrování na jiné view. Tímto způsobem se lze jednoduše pohybovat přes jednotlivé části aplikace. Velkou výhodou tohoto přesměrování je možnost si takto přeposílat parametry, na základě kterých lze poté například zobrazovat určité části stránky apod. Implementace těchto komponent ve zdrojovém kódu je jednoznačná a intuitivní, kromě dodatečných vlastností a stylů, které se dají tlačítkům přiřadit, stačí definovat zobrazovaný text a poté název souboru, jehož obsah se má po stisknutí zobrazit, avšak bez koncové přípony souboru.

Obrázek 3.5 Uspořádání tlačítek pro přesměrování

3.3.3.2 Tabulka

Klasické zobrazení dat ve formě tabulky řeší v PrimeFaces komponenta DataTable. Nabízena je základní tabulka pouze pro zobrazení, ale pomocí vlastností si lze DataTable přizpůsobit podle svých vlastních potřeb. Jedná se například o změnu pořadí sloupců, rozdělení více záznamů na stránky a přepínání mezi nimi, využití mezisoučtových řádků apod. Mezi rozšiřující vlastnosti, které jsem v samotném uvozovacím tagu pro komponentu „dataTable" definoval, patří nastavení možnosti výběru více řádku z tabulky, dále odkaz na proměnnou, do které se takto označené řádky budou v odpovídajícím managed beanu ukládat. Dále následuje záhlaví tabulky, ve kterém se nachází její název, a tlačítko pro ovládání komponenty „columnToggler". Ta poskytuje možnost výběru zobrazovaných sloupců. K tomu, aby docházelo k označování vybraných řádků, je nutno přidat komponentu „growl". Její obsah, tedy i tabulku, lze poté aktualizovat pomocí AJAX událostí, které reagují na výběr řádků nebo zrušení výběru. Následují definice sloupců a jejich odkazování na příslušné atributy v odpovídající managed bean

třídě. Poslední sloupec je zastoupen ikonou lupy a slouží pro editaci řádku, na kterém se ikona nachází. Tato možnost pro úpravu záznamu byla přidána z důvodu potřeby aktualizace informací, pro opravu chybně zadaných hodnot nebo při překlepu apod.



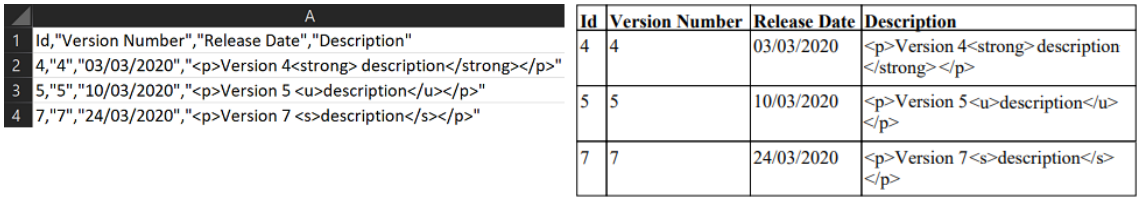
Obrázek 3.6 Tabulka zobrazení Automotive SPICE verzí

```
<p:dataTable id="aspiceVersions" var="aspiceVersionTable" value="#{aspiceVersionView.aspiceVersions}"
    editable="true" selectionMode="multiple" selection="#{aspiceVersionView.selectedAspiceVersions}"
    rowKey="#{aspiceVersionTable.aspiceVersionId}" resizableColumns="true" liveResize="true">
    <f:facet name="header"> ASPICE Versions
        <p:commandButton id="toggler" type="button" value="Columns" style="float:right"
            icon="pi pi-align-justify" />
        <p:columnToggler datasource="aspiceVersions" trigger="toggler" />
    </f:facet>
    <p:ajax event="rowEdit" listener="#{aspiceVersionView.onRowEdit}"
        update=":aspiceVersionForm:aspiceMsgs" />
    <p:ajax event="rowEditCancel" listener="#{aspiceVersionView.onRowCancel}"
        update=":aspiceVersionForm:aspiceMsgs" />
    <p:column headerText="Id" width="6%">
        <h:outputText value="#{aspiceVersionTable.aspiceVersionId}" />
    </p:column>
    <p:column headerText="Version Number" width="14%" style="text-align: center;">
        <h:outputText value="#{aspiceVersionTable.versionNumber}" />
    </p:column>
    <p:column headerText="Release Date" width="12%" style="text-align: center;">
        <h:outputText value="#{aspiceVersionTable.releaseDate}">
            <f:convertDateTime pattern="dd/MM/yyyy"/>
        </h:outputText>
    </p:column>
    <p:column headerText="Description">
        <h:outputText value="#{aspiceVersionTable.description}" escape="false" />
    </p:column>
    <p:column style="width:63px" exportable="false">
         <p:commandButton actionListener="#{aspiceVersionView.edit(aspiceVersionTable)}"
            update=":aspiceVersionForm:addAspiceVersionsDetail :aspiceVersionForm:addaspiceversion
                    :aspiceVersionForm:aspiceVersions"
            oncomplete="PF('addAspiceVersionDialog').show()" image="ui-icon ui-icon-search">
             <f:setPropertyActionListener value="#{aspiceVersionTable}"
                 target="#{aspiceVersionView.selectedAspiceVersion}" />
         </p:commandButton>
     </p:column>
</p:dataTable>
```

Výpis 3.2 XHTML struktura tabulky pro zobrazeni ASPICE verzí

Po kliknutí na již zmiňovanou lupu se otevře dialogové okno, které je stejné jako v případě přidávání, ale aby odpovídalo tomu, že se jedná o úpravu existujícího záznamu, se mu dynamicky mění v záhlaví název a text potvrzujícího tlačítka vzhledem k prováděné činnosti. Všechny komponenty určené pro uživatelský vstup se předvyplní hodnotami vybraného záznamu. Detailní popis zabývající se těmito dialogy je popsán v následující kapitole 3.3.3.3. V zápatí

tabulky jsem poté kromě tlačítek pro přidání, smazání a zobrazení detailu vybraných záznamů, popsaných v dalších kapitolách, přidal i tlačítka umožňující export zobrazené tabulky do zvoleného formátu. U možnosti exportování zobrazených dat v tabulce je možno vybírat z několika podporovaných typů výstupních souborů, avšak jsem se rozhodl pro přidání pouze dvou nejčastěji používaných – soubory typu CSV a PDF. V obou případech jsem komponentám „dataExporter", zprostředkovávající stažení dokumentů, nastavil vlastnost „selectionOnly", aby si uživatel mohl zvolit, které řádky z tabulky chce mít ve svém konečném výstupu. Jeho formát se dá do detailu přizpůsobit přetypováním objektu dokumentu na třídu „Document". Takto lze změnit typ formátu, barvu pozadí, přidat logo a další.



	A
1	Id,"Version Number","Release Date","Description"
2	4,"4","03/03/2020"," Version 4 description "
3	5,"5","10/03/2020"," Version 5 <u>description</u> "
4	7,"7","24/03/2020"," Version 7 <s>description</s> "

Id	Version Number	Release Date	Description
4	4	03/03/2020	Version 4description
5	5	10/03/2020	Version 5<u>description</u>
7	7	24/03/2020	Version 7<s>description</s>

Obrázek 3.7 Formát exportovaných dat tabulky v CSV a PDF souborech

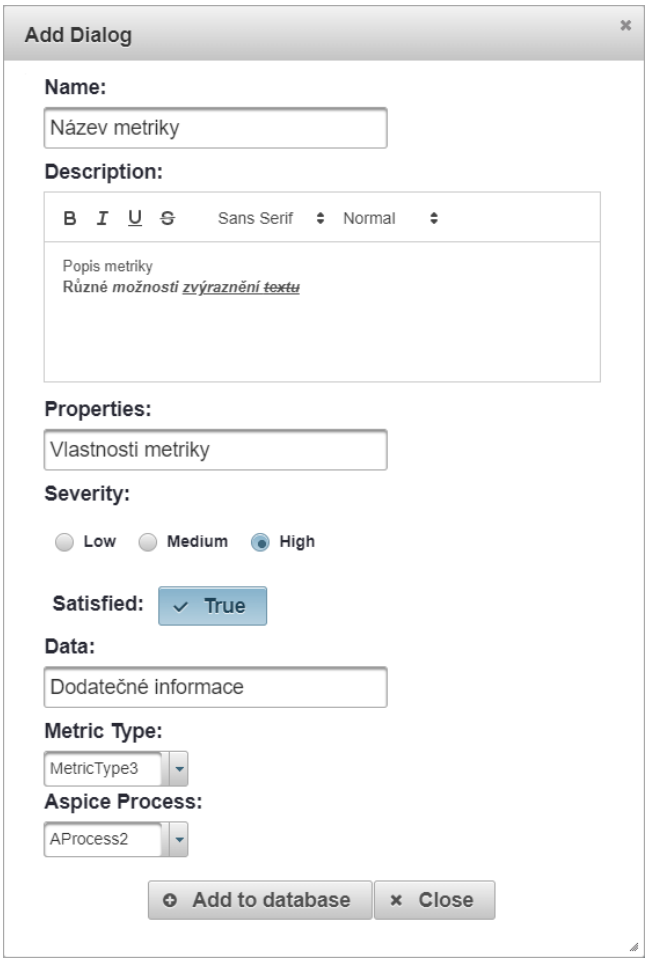
```
<f:facet name="footer">
    <p:commandButton process="aspiceVersions" icon="pi pi-plus-circle" value="Add Aspice Version"
        update=":aspiceVersionForm:aspiceVersions :aspiceVersionForm:addaspiceversion"
        action="#{aspiceVersionView.clearSelected}" oncomplete="PF('addAspiceVersionDialog').show()"
        actionListener="#{aspiceVersionView.changeHeaderAndButtonInAddDialog('Add Dialog', 'Add to database')}"/>
    <p:commandButton process="aspiceVersions" update=":aspiceVersionForm:multiAspiceVersionsDetail"
        icon="pi pi-search" value="View" oncomplete="PF('multiAspiceVersionsDialog').show()" />
    <p:commandButton process="aspiceVersions" update=":aspiceVersionForm:deleteAspiceVersionsDetail"
        icon="pi pi-times" value="Delete" oncomplete="PF('multiDeleteDialog').show()" />
    <div style="float:right;padding-top: 5px;">
        <h:commandLink>
            <p:commandButton value="PDF" width="24"/>
            <p:dataExporter type="pdf" target="aspiceVersions" fileName="aspiceVersions" selectionOnly="true"/>
        </h:commandLink>
        <h:commandLink>
            <p:commandButton value="CSV" width="24"/>
            <p:dataExporter type="csv" target="aspiceVersions" fileName="aspiceVersions" selectionOnly="true"/>
        </h:commandLink>
    </div>
</f:facet>
```

Výpis 3.3 Zápatí tabulky ASPICE verzí s tlačítky pro manipulaci se záznamy a export dat

3.3.3.3 *Dialogová okna*

V této kapitole, která se zabývá popisem průběhu vytváření dialogových oken, se zaměřuji přímo na popis entity metrik, protože má z hlediska datového modelu nejvíce návazností, a tvorba dialogů vyžadovala použití více komponent. Práci na dialogových oknech jsme si rozdělili v týmu tak, aby každý pracoval na jiné části z datového modelu, a následně jsme naše části pomocí využití verzovacího systému Git spojili. Do databáze se dalo zatím přidávat data pouze pomocí SQL Editoru integrovaného v programu DBeaver, ke kterému by běžný uživatel neměl přístup, a tak jsem musel jsme pro jednotlivé tabulky přidat možnost přidávat nové záznamy. Pod tabulkou se vyskytuje tlačítko, po jehož stisknutí se zobrazí přidávací dialog realizovaný stejnojmennou komponentou z PrimeFaces nabídky. Ten se skládá z několika vstupních polí pro zapisování hodnot, dále v případě entity Metric obsahuje i přepínače pro výběr z možností pro nastavení proměnné „Severity". Sloupec „Satisfied" nabývá pouze dvou hodnot,

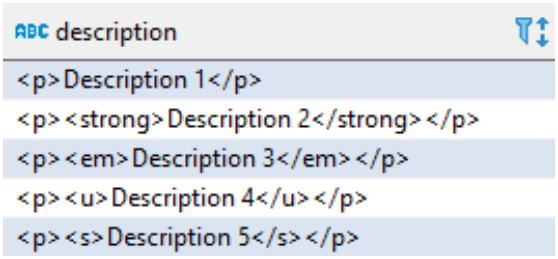
proto jsem v tomto případě volil možnost přepínacího tlačítka. Pole pro „Description" není řešeno jako v ostatních případech pouze volným řádkem pro psaní, ale jeden z požadavků byl, aby uživatel mohl při vyplňování popisu položky zvýrazňovat důležité části textu apod. V nabídce PrimeFaces komponent jsem našel odpovídající prvek „TextEditor", který poskytuje všechny klasické možnosti úpravy textu. Do jeho záhlaví lze přidat libovolná ovládací tlačítka pro změnu typu písma – tučnost, kurzíva, podtržení, přeškrtnutí aj., přičemž jejich výběr a umístění lze jednoduše změnit.



Obrázek 3.8 Dialogové okno pro přidání nové metriky

Z uvedeného obrázku zobrazených exportovaných dat (Obrázek 3.7) v kapitole 3.3.3.2, si lze všimnout, jakým způsobem tento použitý textový editor ukládá vybrané stylizování písma. Využívá k tomu HTML značek, které se ukládají společně s daným textem. Z důvodu použití těchto značek, které mohou obsahovat mnoho znaků, jsem musel v databázovém systému pomocí DBeaveru změnit datový typ uložených dat sloupce sloužícího pro popis entit na „longtext". Do něho lze uložit až $2^{32} - 1$ znaků [38]. Aby se poté po zobrazení takto uložených dat nevyskytovaly v textu tyto tagy a dané stylizování se na text správně aplikovalo, poskytuje pro tento případ knihovna PrimeFaces u prvků grafického rozhraní určených pro zobrazení textu na výstupu atribut „escape". Ten je ve výchozím stavu nastaven na „True" a označuje, že s ním

spojená komponenta obsahuje tzv. „escape characters", které se při nastavení hodnoty na „False" interpretují jako HTML nebo XML značky.



Obrázek 3.9 Reprezentace stylizace textu

V zápatí dialogu se poté nachází tlačítko pro přidání záznamu z vyplněných hodnot. Při jeho stisknutí se zavolá metoda „addMetric", která se nachází v odpovídajícím managed beanu. Protože se tato funkce volá i při editování metriky, což je popsáno dále v této kapitole, je nutno na začátku volání rozlišit, o kterou akci se jedná. Editace proběhne, pouze pokud je už nějaký řádek z tabulky vybrán, v tom případě se ještě zkontroluje, jestli metrika s vybraným ID existuje a pokud ano, dojde k předvyplnění uložených parametrů a záznam se aktualizuje. Jestliže není žádný řádek vybraný, jedná se o přidání nových dat. Musí se vytvořit nový objekt metriky, nastavit mu parametry a přidat do databáze.

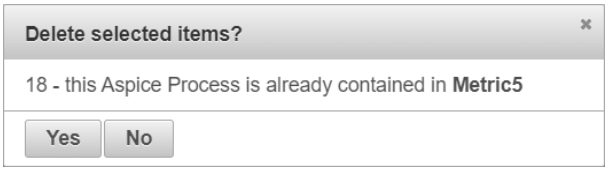
```
public void addMetric() {
    if (selectedMetric != null) {
        if (metricExists(selectedMetric.getMetricId())) {
            selectedMetric.setName(name);
            selectedMetric.setDescription(description);
            selectedMetric.setProperties(properties);
            selectedMetric.setData(data);
            selectedMetric.setSeverity(severity);
            selectedMetric.setSatisfied(satisfied);
            selectedMetric.setMetricTypeId(getMetricTypeIdByName(metricTypeName));
            selectedMetric.setAspiceProcessId(getAspiceProcessIdByName(aspiceProcessName));
            MetricLocalServiceUtil.updateMetric(selectedMetric);
        }
    } else {
        metric = MetricLocalServiceUtil.createMetric(0);
        metric.setName(name);
        metric.setDescription(description);
        metric.setProperties(properties);
        metric.setSeverity(severity);
        metric.setSatisfied(satisfied);
        metric.setData(data);
        metric.setMetricTypeId(getMetricTypeIdByName(metricTypeName));
        metric.setAspiceProcessId(getAspiceProcessIdByName(aspiceProcessName));
        MetricLocalServiceUtil.addMetric(metric);
    }
    metrics = MetricLocalServiceUtil.getAllMetrics();
}
```

Výpis 3.4 Metoda addMetric pro přidání nebo úpravu metrik

Kromě přidávacího tlačítka, jehož proces fungování je popsán výše, se v zápatí dialogu ještě vyskytuje další tlačítko pro jeho zavření a vymazání vyplněných hodnot pro příští použití. Mimo klíčová slova, popisující činnosti těchto tlačítek, jsem přidal z nabídky grafických elementů i ikonky, pro jejichž použití bylo nutno přidat do projektu odpovídající závislosti. Dále jsem

pro pohodlné a efektivní používání nastavil stejnou funkcionalitu, jako má zavírací tlačítko, pro fyzickou klávesu na klávesnici „Escape".

Další základní manipulací se záznamy uloženými v databázi je jejich smazání. K tomu se využívá již dříve zmíněného označení řádku tabulky. Po kliknutí do volného místa tabulky se vybere odpovídající řádek, také lze poté držet klávesu „Shift" a kliknout na jiný záznam, tím dojde k označení všech záznamů mezi těmi vybranými, nebo s pomocí klávesy „Ctrl" lze vybrat jednotlivé záznamy ručně. Pod tabulku, vedle tlačítka pro zobrazení přidávajícího dialogu, jsem tedy pro každou tabulku přidal tlačítko pro smazání označených údajů. V těle dialogu se zobrazí identifikátory označených záznamů a jejich další informace. Pokud se uživatel chystá smazat záznam, který je právě používán už v jiné tabulce, zobrazí se uživateli zpráva s upozorněním této události. Součástí této zprávy jsou tučně zvýrazněny názvy všech záznamů spojených s tím, který se uživatel chystá smazat.



Obrázek 3.10 Dialog smazání záznamu s upozorněním

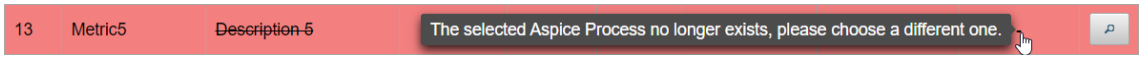
Po stisknutí tlačítka pro potvrzení se záznamy z databáze, a tedy i z tabulky, odeberou a komponenta DataTable se pro zobrazení provedené operace obnoví díky vlastnosti „update" s nastaveným parametrem. Hodnota tohoto aktualizujícího parametru se zadává do uvozovek a je nutno postupně zadat identifikátory jednotlivých komponent, do kterých je ta s požadovanou aktualizací vnořená, včetně jejího ID. Jednotlivé části cesty se oddělují symbolem dvojtečky, která se musí vyskytovat i na začátku parametru. Stejně jako u předchozího případu přidání, má i dialog pro smazání tlačítko pro zrušení prováděné operace a zavření okna. Pokud by nastal předchozí případ, při kterém by uživatel smazal záznam, na který byla vytvořená závislost z jiné tabulky, je nutné, aby byl o této události informován. K tomu jsem použil vlastnosti tabulky „rowStyleClass", díky které lze na řádky aplikovat různé CSS styly. Pokud tedy došlo k smazání předtím použité hodnoty, aplikuje se pomocí ternárního operátoru na ovlivněný řádek s chybějícím údajem uvedený vytvořený styl. V opačném případě se pro zobrazení volí výchozí nastavení tabulky.

```
.error {
    background-color: rgba(231,0,0,0.50) !important;
    background-image: none !important;
    color: #000000 !important;
}
```

Výpis 3.5 CSS styl pro zvýraznění řádku

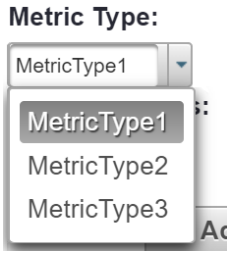
Barvu pozadí řádku jsem volil z ověřených barev, které jsou všeobecně používané pro chybové záležitosti, a nastavil jsem jí 50% průhlednost. Aby se šedý text v červeném pozadí neztrácel, ba naopak vynikal, změnil jsem ještě barvu textu na černou. Za nastavené vlastnosti je nutno přidat klíčové slovo „important" se symbolem vykřičníku na začátku, aby došlo k přepsání výchozího stylu komponenty. Namísto smazané hodnoty jsem zvolil zobrazení symbolu pomlčky označující chybějící záznam a při přejetí myší přes tento symbol je uživateli zobrazena zpráva,

která popisuje, že došlo ke smazání závislého záznamu a že je nutno vybrat jiný. Tuto vyskakovací zprávu řeší komponenta „tooltip".



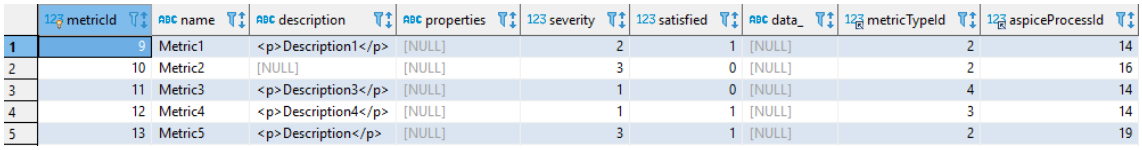
Obrázek 3.11 Zvýraznění řádku s chybějící závislostí

V úvodní části kapitoly 3.2 a v průběhu této kapitoly jsem popisoval jednotlivé návaznosti jedné entity na druhou a naopak. Jestliže má tedy entita představující Metriky návaznost na MetricType a zároveň na ASPICE Process, bylo nutné tato propojení při vytváření metrik zohlednit. V přidávacím dialogovém okně jsem tedy přidal komponentu „SelectOneMenu", jejíž položky jsou tvořeny z názvů vytvořených typů metrik nebo procesů ASPICE.



Obrázek 3.12 Výběr z vytvořených typů metrik

Protože v databázové vrstvě je návaznost mezi entitami pouze na základě identifikátorů, a pro uživatele by mohlo být velice nepraktické a nepohodlné si pamatovat typy metrik a ASPICE procesy podle jejich ID, musel jsem v odpovídající managed bean třídě vytvořit metody pro získání všech názvů z vytvořených typů metrik a všech čísel verzí procesů. Takto vytvořené listy stačilo poté použít jako návratové hodnoty pro výběrové menu a problém s propojením pomocí pouze uložených identifikátorů byl vyřešen. Tímto způsobem jsme každý z týmu vyřešili pro všechny existující návaznosti, tedy ještě mezi ASPICE verzemi a jim odpovídajícím ASPICE procesům. Naimplementovat řešení výše zmíněného problému bylo náročné, protože uživatel pracuje pouze s reprezentujícími jmény typů metrik nebo čísly verzí, ale v pozadí se vše řídí podle identifikátorů. Musel jsem si průběžně při práci kontrolovat, zda hodnoty, se kterými jsem pracoval byly ty správné, aby nedocházelo k chybnému přiřazení. K této kontrole jsem používal program DBeaver (viz 2.10), který bylo nejprve nutno propojit s již běžící databází od MariaDB. K tomu sloužil velice intuitivní a jednoduchý průvodce v aplikaci, díky kterému se vše potřebné nastavilo. Zabudovaný SQL editor sloužil při analýze správnosti ukládání dat pro jednoduché přidávání a upravování testovaných dat.



	metricId	name	description	properties	severity	satisfied	data_	metricTypeId	aspiceProcessId
1	9	Metric1	Description1	[NULL]	2	1	[NULL]	2	14
2	10	Metric2	[NULL]	[NULL]	3	0	[NULL]	2	16
3	11	Metric3	Description3	[NULL]	1	0	[NULL]	4	14
4	12	Metric4	Description4	[NULL]	1	1	[NULL]	3	14
5	13	Metric5	Description	[NULL]	3	1	[NULL]	2	19

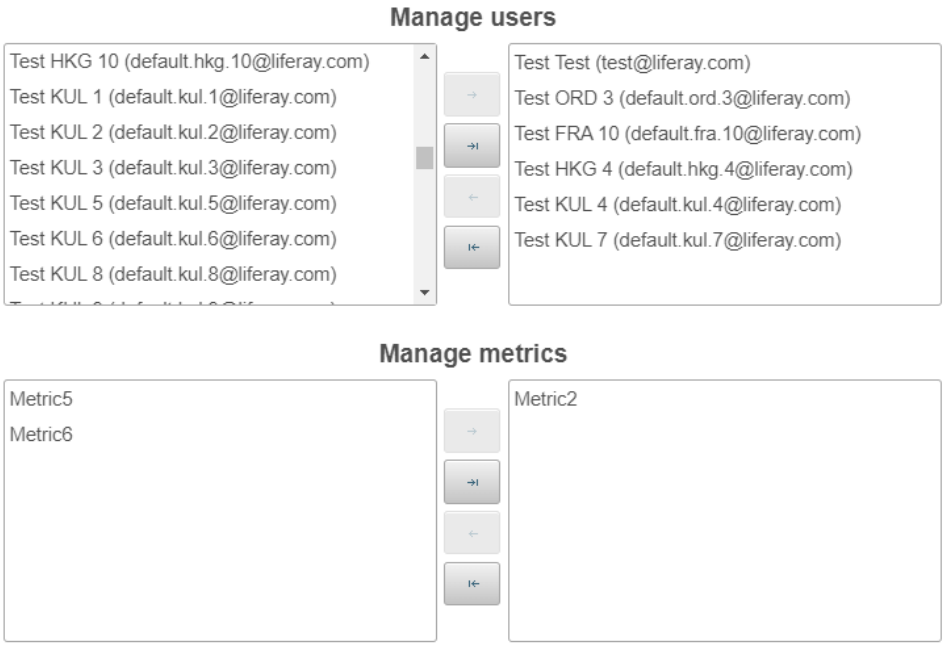
Obrázek 3.13 Reprezentace dat programem DBeaver

3.3.4 Využití Dynamic Query

Pro snadný přístup k datům uloženým v databázi poskytuje Liferay metody, které jsou naimplementovány v „LocalServiceUtil" třídách, zastupující jednoduché SQL dotazy. Jedná se například o získání všech záznamů podle různých kritérií, jejich smazání, přidání nebo úprava hodnot. Avšak někdy, jak tomu bylo i v případě tohoto systému, je potřeba konstruovat specifičtější dotazy. Pro tuto realizaci Liferay portál umožňuje využití tzv. dynamických dotazů, které pracují na základě Hibernate frameworku — nástroje pro mapování relačních objektů s použitím programovacího jazyku Java. Z této definice je patrné, že pro konstrukci dynamických dotazů není nutná znalost SQL jazyka, což bylo pro mě velmi přínosné, jelikož jsem se databázové systémy učil pouze na střední škole.

Dynamic Query jsem využil převážně při implementaci entity „Project". Ta podle datového modelu (Obrázek 3.3) má návaznost na uživatele a metriky. V obou případech se jedná o vazbu typu M:N, která naznačuje, že součástí projektu může být více uživatelů, a zároveň jeden uživatel může být součástí více projektů. Toto tvrzení platí analogicky pro vztah mezi metrikami a projekty. Takto mezi nimi vzniknou vazební entity „ProjectUser" a „ProjectMetric", které zastupují konkrétní případy propojení. Za úkol jsem dostal vytvořit stránku umožňující práci s projekty. Pro zobrazení projektu jsem vybral grafický prvek „DataView", který svým vzhledem i použitím připomíná již dříve použitou a popsanou tabulku. Každý jednotlivý řádek této komponenty je zastoupen jedním projektem definovaným svým jménem, stylizovaným popisem, a datem založení. Pro vytvoření vlastního projektu slouží přidávací tlačítko, které otevírá dialogové okno se vstupními poli pro zadávání zmíněných údajů projektu. Po potvrzení vytvoření projektu se zadanými hodnotami se vytvoří odpovídající záznam v databázi o samotném projektu, a rovněž vznikne nový vztah mezi takto vytvořeným projektem, a právě přihlášeným uživatelem provádějícím tuto činnost — projektový uživatel. Ten je tvořen pouze identifikačními čísly dvou entit, ze kterých vychází. Pro zobrazení detailu každého projektu slouží tlačítko zastoupené symbolem lupy, po jehož stisknutí se otevře dialogové okno obsahující základní informace o projektu, a navíc dvě „PickList" komponenty. Jedná se o grafické znázornění seznamů hodnot ve formě dvou sloupců. Obsah levého sloupce je tvořen všemi hodnotami, ze kterých může uživatel pomocí ovládacích prvků umístěných uprostřed mezi sloupci vybírat. Takto vybrané hodnoty se poté přesunují do druhého sloupce umístěného v pravé části. Tuto komponentu jsem vybral kvůli její jednoduchosti a jednoznačnosti používání pro přidávání uživatelů, kteří se budou na projektu podílet a pro výběr metrik, které bude projekt sledovat. Aby došlo ke správnému naplnění obou listů, i v případě opětovného otevření dialogu s detailem projektu, bylo nutné předvyplnit všechny sloupce podle uložených záznamů v databázi. Hodnoty pravého sloupce s cílovými daty jsem vytvořil díky vzniklým vazebním entitám, ve kterých jsem podle identifikátoru projektu vyhledával všechny odpovídající uživatele a metriky. Následovalo získání obsahu pro levý sloupec, pro které už bylo nutno, abych využil dynamických dotazů. Při vytvoření objektu třídy „Dynamic Query" jsem musel definovat, které třídy (entity) se tento dotaz týkal. V případě prvního seznamu jde o uživatele, které sám implementuje Liferay Portal pomocí třídy „User". Dále jsem musel pomocí třídy „PropertyFactoryUtil" specifikovat podmínku, podle které se v databázi vyhledává — podle jedinečného identifikátoru uživatele. Podmínka mimo to ještě

vyžaduje seznam hodnot zadaných ve formě pole. Pomocí jednoduchého dotazu jsem uložil do proměnné všechny vytvořené uživatele a z tohoto listu jsem odstranil všechny projektové uživatele. Výsledný list vzniklý touto rozdílovou operací stačilo použít jako pole hodnot pro parametr podmínky a získat tak záznamy pro levý sloupec komponenty. Analogicky postup platil i pro naplnění druhého listu reprezentujícího metriky.



Obrázek 3.14 Výběr projektových uživatelů a metrik

```
DynamicQuery getSourceUsers = DynamicQueryFactoryUtil
        .forClass(User.class, User.class.getClassLoader())
        .add(PropertyFactoryUtil.forName("userId").in(allUsersIds.toArray()));
usersSource = UserLocalServiceUtil.dynamicQuery(getSourceUsers);
```

Výpis 3.6 Dynamic Query pro získání uživatelů

Jak jsem dříve v této kapitole zmínil, tak obsahy sloupců komponenty „PickList" se tvoří na základě listů uživatelů nebo metrik. Protože například jednotliví uživatelé mají v databázi uložené veškeré kontaktní informace, IP adresu posledního přihlášení, ale i zašifrované heslo a odpověď na kontrolní otázku v případě ztráty tohoto hesla, bylo nutno zobrazovaný výstup upravit. Naštěstí použitá komponenta k tomu disponuje vlastností „itemLabel", která reprezentuje titulek, který se zobrazuje. V případě uživatelů jsem tedy pro zobrazení vybral kombinaci jména a příjmení, které přímo poskytuje metoda „getFullName", a pro reprezentaci jednotlivých metrik jsem vybral jejich názvy. Přihlášený uživatel tedy může přidávat a odebírat ostatní uživatele do projektu (kromě sebe samotného) a manipulovat s předem vytvořenými metrikami. Při potvrzení svého výběru tlačítkem dochází k uložení dat do databáze. Tato situace se dala v rámci tohoto projektu řešit více způsoby. Jedním z nich bylo smazání všech dosavadních odpovídajících uložených záznamů a jejich opětovné přidání. Toto první řešení je velmi jednoduché na implementaci, a navíc práce s databází je efektivní, avšak v případě, kdyby se přidával pouze jeden uživatel, musela by se všechna předchozí propojení smazat a vytvořit nová. Mimo to by

tento způsob nebylo možno použít v případě metrik, jelikož jeden z atributů projektových metrik je datum přidání, a tak by docházelo k neustálému přepisování této hodnoty. Proto jsem se rozhodl pro druhý způsob, který je velmi podobný procesu inicializace hodnot. Nejprve se porovnají všechny uložené záznamy s těmi, které se uživatel chystá změnit. Pokud se některé z těchto již dříve vytvořených záznamů z databáze neshodují s těmi novými, které se uživatel chystá uložit, dojde k jejich odstranění. V opačném případě, kdy se některý z nových údajů v databázi ještě nevyskytuje, se vytvoří nové spojení.

```
DynamicQuery getProjectUsers = ProjectUserLocalServiceUtil.dynamicQuery()
        .add(PropertyFactoryUtil.forName("projectID").eq(projectId));
List<ProjectUser> projectUsers = ProjectUserLocalServiceUtil.dynamicQuery(getProjectUsers);

List<Long> usersTargetIds = new ArrayList<>();
for (User u : usersTarget) {
    usersTargetIds.add(u.getUserId());
}
if (!usersTargetIds.contains(userId)) {
    usersTargetIds.add(userId);
}
List<Long> projectUserIds = new ArrayList<Long>();
for (ProjectUser pu : projectUsers) {
    projectUserIds.add(pu.getUserID());
    if (!usersTargetIds.contains(pu.getUserID())) {
        ProjectUserLocalServiceUtil.deleteProjectUser(pu);
    }
}
for (User u : usersTarget) {
    if (!projectUserIds.contains(u.getUserId())) {
        projectUser.setProjectID(projectId);
        projectUser.setUserID(u.getUserId());
        ProjectUserLocalServiceUtil.addProjectUser(projectUser);
    }
}
```

Výpis 3.7 Vytváření nových a odebírání starých projektových uživatelů

Entity projektových uživatelů jsem dále například využil při načítání stránky s projekty. Nejprve se pomocí Liferay nástrojů zjistí identifikátor právě přihlášeného uživatele a podle něho vytvoří dynamický dotaz, jehož cílem je získat z tabulky projektových uživatelů seznam identifikátorů všech projektů, které jsou spojeny s přihlášeným uživatelem. Na základě tohoto seznamu se poté získá dalším dynamickým dotazem výsledný list projektů, které se uživateli na stránce zobrazí.

Závěr

Cílem této bakalářské práce bylo popsat průběh mého absolvování odborné praxe ve firmě SCOVECO, s.r.o., během které jsem byl součástí vývoje systému pro sledování a zobrazování metrik tak, aby splňovaly standard Automotive SPICE. V rámci své praxe jsem se podílel na vytvoření funkčního systému, jehož funkčnosti, které jsem implementoval, jsou popsány v této bakalářské práci.

V průběhu této odborné praxe jsem využil řadu dovedností a znalostí, které jsem měl možnost získat během studia. Díky předmětu Úvod do softwarového inženýrství jsem při analýze systému využil předchozích zkušeností se zaznamenáním funkčností systému pomocí UML diagramu, podle kterého se následně vytvořil datový model celého systému. Tím se rozsáhlý projekt rozdělil na menší části, jejichž funkcionalitu jsem poté pomocí předchozího osvojení programování z řady zaměřených předmětů mohl postupně zpracovávat. Při návrhu uživatelského rozhraní jsem využil znalostí struktury pro tvorbu webových stránek.

Z počátku bylo zapotřebí, abych se seznámil se samotnými teoretickými pojmy zadání praxe a také s novými technologiemi, které byly při vývoji systému použity. V celém průběhu bylo zapotřebí průběžně používat verzovací systém Git, s kterým jsem měl doposud pouze základní zkušenosti, a jehož použití značně zjednodušilo a urychlilo práci v týmu. V té jsem se společně s komunikací mohl absolvováním této odborné praxe zlepšit. Značnou část jsem musel věnovat studiem a následně prací s databázovými systémy, se kterými jsem kvůli volbě svého oboru neměl s výjimkou střední školy předchozí zkušenosti. Bylo nutno, abych se k těmto základním znalostem doučil i řadu nových. Velkou částí bylo také seznámení a následné využití nástrojů poskytovaných Liferay portálem, díky kterým se vývoj oproti ručnímu zpracování značně zjednodušil. Rovněž jsem získal cenné zkušenosti v oblasti vytváření uživatelského rozhraní pomocí komponent s použitím grafické knihovny PrimeFaces.

Při tvorbě uživatelského rozhraní webové aplikace se příliš nezabývalo jeho stylizací, jelikož se předpokládá použití jednotného grafického stylu z nabídky vývojářů PrimeFaces pro celý portál Liferay, a navíc hlavní zaměření bylo na zpracování funkčnosti systému. Mezi další způsoby rozšíření systému lze například zmínit přidání a zařazení uživatelů na základě přístupových rolí v rámci projektu, kterého jsou součástí, a další.

Z mého pohledu hodnotím absolvování odborné praxe za velmi pozitivní a přínosné, jelikož jsem měl možnost podílet se rozsáhlém projektu s využitím mnoha pokročilých technologií, a tak získat řadu cenných praktických zkušeností. Těch jsem rovněž nabyl od zkušených programátorů, kteří mi v celém průběhu byli ve všem vstřícní a nápomocni.

Použitá literatura

[1] SCOVECO, s.r.o. [online]. [cit. 2020-04-24]. Dostupné z: https://scoveco.cz/cs/

[2] VONDRÁK, Ivo. Úvod do softwarového inženýrství [online]. Ostrava, 2002 [cit. 2020-04-01]. Dostupné z: http://vondrak.cs.vsb.cz/download/Uvod_do_softwaroveho_inzenyrstvi.pdf. VŠB – Technická univerzita Ostrava Fakulta elektrotechniky a informatiky katedra informatiky.

[3] Software Engineering Metrics: What Do They Measure and How Do We Know [online]. 2004 [cit. 2020-04-02]. Dostupné z: http://www.kaner.com/pdfs/metrics2004.pdf

[4] ISO – International Organization for Standardization: ISO/IEC 15504 [online]. 2012 [cit. 2020-03-17]. Dostupné z: https://www.iso.org/standard/60555.html

[5] Automotive SPICE Process Assessment / Reference Model [online]. 2017 [cit. 2020-03-17]. Dostupné z: http://www.automotivespice.com/fileadmin/software-download/AutomotiveSPICE_PAM_31.pdf

[6] Automotive SPICE: Process Reference Model [online]. Automotive SIG, 2010 [cit. 2020-03-31]. Dostupné z: http://www.automotivespice.com/fileadmin/software-download/automotiveSIG_PRM_v45.pdf

[7] Automotive SPICE: Process Reference Model [online]. VDA QMC Working Group 13 / Automotive SIG, 2017 [cit. 2020-03-31]. Dostupné z: http://www.automotivespice.com/fileadmin/software-download/AutomotiveSPICE_PAM_31.pdf

[8] Java JDK, JRE and JVM [online]. [cit. 2020-03-23]. Dostupné z: https://www.programiz.com/java-programming/jvm-jre-jdk

[9] What is Java programming language? [online]. [cit. 2020-03-23]. Dostupné z: https://howtodoinjava.com/java/basics/what-is-java-programming-language/

[10] Java Programming Language [online]. [cit. 2020-03-23]. Dostupné z: https://web.mit.edu/java_v1.5.0_22/distrib/share/docs/guide/language/index.html

[11] JavaServer Faces [online]. [cit. 2020-03-23]. Dostupné z: http://www.javaserverfaces.org/

[12] JSF – Managed Beans [online]. [cit. 2020-03-23]. Dostupné z: https://www.tutorialspoint.com/jsf/jsf_managed_beans.htm

[13] Help – Eclipse Platform: Perspectives [online]. [cit. 2020-03-18]. Dostupné z: https://help.eclipse.org/2019-12/index.jsp?topic=%2Forg.eclipse.platform.doc.user%2Fconcepts%2Fconcepts-4.htm

[14] Version Management [online]. [cit. 2020-03-23]. Dostupné z: https://its.unl.edu/bestpractices/version-management

[15] Git: About [online]. [cit. 2020-03-18]. Dostupné z: https://git-scm.com/about

[16] Git: About Version Control [online]. [cit. 2020-03-18]. Dostupné z: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

[17] GitLab: The DevOps Lifecycle with GitLab [online]. [cit. 2020-03-17]. Dostupné z: https://about.gitlab.com/stages-devops-lifecycle/

[18] Gradle User Manual. Gradle [online]. 2019 [cit. 2019-12-28]. Dostupné z: https://docs.gradle.org/current/userguide/userguide.html

[19] Liferay Portal. Liferay Community [online]. 2019 [cit. 2019-12-28]. Dostupné z: https://liferay.dev/-/portal

[20] Liferay Portal: What is Service Builder? [online]. [cit. 2020-03-15]. Dostupné z: https://portal.liferay.dev/docs/7-1/tutorials/-/knowledge_base/t/what-is-service-builder

[21] The MIT License [online]. [cit. 2020-03-15]. Dostupné z: https://opensource.org/licenses/MIT

[22] PrimeFaces [online]. PrimeTek Informatics, 2019 [cit. 2020-03-15]. Dostupné z: https://www.primefaces.org/showcase/

[23] PrimeFaces Showcase: Timeline [online]. PrimeTek Informatics, 2019 [cit. 2020-04-02]. Dostupné z: https://www.primefaces.org/showcase/ui/data/timeline/basic.xhtml

[24] PrimeFaces: Support [online]. [cit. 2020-03-21]. Dostupné z: https://www.primefaces.org/support/

[25] Web Application Life Cycle [online]. [cit. 2020-03-21]. Dostupné z: https://pirlwww.lpl.arizona.edu/resources/guide/software/jwsdp/tutorial/doc/WebApp2.html

[26] Apache License, Version 2.0 [online]. [cit. 2020-03-21]. Dostupné z: https://www.apache.org/licenses/LICENSE-2.0

[27] GitHub – Apache Tomcat [online]. [cit. 2020-03-21]. Dostupné z: https://github.com/apache/tomcat

[28] About MariaDB Server [online]. [cit. 2020-03-19]. Dostupné z: https://mariadb.org/about/

[29] MariaDB Knowledge Base [online]. [cit. 2020-03-19]. Dostupné z: https://mariadb.com/kb/en/about-mariadb-software/

[30] MySQL Performance: MySQL vs. MariaDB [online]. [cit. 2020-03-19]. Dostupné z: https://www.liquidweb.com/kb/mysql-performance-mysql-vs-mariadb/

[31] Microsoft JDBC Driver for SQL Server [online]. 2019 [cit. 2020-03-23]. Dostupné z: https://docs.microsoft.com/en-us/sql/connect/jdbc/microsoft-jdbc-driver-for-sql-server?view=sql-server-ver15

[32] About DBeaver [online]. 2019 [cit. 2020-03-23]. Dostupné z: https://github.com/dbeaver/dbeaver/wiki

[33] The Unified Modeling Language [online]. [cit. 2020-04-03]. Dostupné z: https://www.uml-diagrams.org/

[34] GOTHE, Deepak. Understanding the Java Portlet Specification 2.0 (JSR 286) [online]. leden 2010 [cit. 2020-04-30]. Dostupné z: https://www.oracle.com/technetwork/java/jsr286-141866.html

[35] Portlets [online]. [cit. 2020-04-04]. Dostupné z: https://portal.liferay.dev/docs/7-0/tutorials/-/knowledge_base/t/portlets

[36] CamelCase definition [online]. Sharpened Productions, 2019 [cit. 2020-03-19]. Dostupné z: https://techterms.com/definition/camelcase

[37] Defining Service Entity Finder Methods [online]. [cit. 2020-04-30]. Dostupné z: https://help.liferay.com/hc/en-us/articles/360017886632-Defining-Service-Entity-Finder-Methods

[38] LONGTEXT [online]. [cit. 2020-03-29]. Dostupné z: https://mariadb.com/kb/en/longtext/